Virtual Environments for Training

Quarterly Status Report

December 22, 1995 to April 5, 1996

Submitted by R. Stiles
Lockheed Martin Advanced Technology Center

Prepared for
Office of Naval Research
Contract N00014-95-C-0179
April 1996
(CDRL Report A001 - Progress Report)

Abstract: This report describes the Lockheed-Martin VET team efforts and accomplishments during the second quarter of the Virtual Environments for Training contract, awarded September 22, 1995. Contract activity is reported in six major areas: Software Design, Software Development, Contract Administration and Resources, Contract Review, Domain Selection/Development, and Conferences and Workshops. This report contains significant material submitted to Lockheed Martin by Dr. Allen Munro at USC/BTL and Dr. Lewis Johnson at USC/ISI.

Table of Contents

Summary

This quarterly report discusses activity for the second 3 months of the Virtual Environments for Training (VET) contract. Efforts continued on tasks initiated and detailed in the first quarterly report (dated January 1996). The VET development team's effort for selection of the domain that will be used in the first demonstration was concluded. Much effort during the second quarter was devoted to integration of the 3 main software elements, Vista, RIDES, and SOAR. Communication protocols were designed and developed, and a series of integration sessions using the High Pressure Air Compressor domain indicate successful communications between each of the systems.

During this quarter, we developed an integrated planning architecture for Soar agents, which can be used to enable agents to demonstrate tasks, monitor students performing tasks, and explaining how to perform tasks. Authoring capabilities were developed to facilitate the specification of tasks, both to drive the performance of these tasks by agents and the training of individuals in performing these tasks. We developed key communication mechanisms enabling Soar agents to operate on virtual worlds implemented using Vista and V-RIDES, and allowing them to control human figures modeled using Jack. The Debrief system for explaining the actions of Soar agents was adapted for use within VET. Speech generation capability for the STEVE system was started and is underway. Thus key infrastructure is in place for developing pedagogical agents in terms of communicating with the training simulations, the virtual environment, and the students.

Early in the second quarter, subcontracts with both the USC Behavioral Technology Laboratories and USC Information Sciences Institute were finalized. By the end of this second quarter, all three contract groups had received the majority of the hardware and software ordered during the first quarter and a regular mechanism for invoicing the Office of Naval Research every two weeks was put in place.

Introduction

This report covers the second quarter in the four quarters that comprise system development for the Training Studio software. In this document we discuss the design and development of extensions to the initial system, integration of the three separate process comprising the Training Studio, our survey of advanced practice in the technical field, efforts involving the sponsoring agency in development choices, and final selection of the team's recommendation for an initial applications domain to guide initial system development.

The four main organizations contributing to the VET effort are the Lockheed Martin Advanced Technology Center, the USC Behavioral Technology Laboratories, the USC Information Sciences Institute, and the USAF Technical Training Research Division. Because we are undergoing initial development, the effort of the Technical Training Research Division, whose contribution is the evaluation of the system with students and for ease of use, are not covered.

Knowing the roles of the contributing organizations is important to understanding how the activities described in this status report contribute toward the completed system.

For system development, the roles of the organizations are clearly delineated and complementary. The Information Sciences Institute is concerned with modeling the human in the 3D training environment, the Behavioral Technology Laboratories is concerned with object simulation tied to instruction, and the Lockheed Martin Advanced Technology Center is

concerned with real-time display and interaction in a virtual environment, as well as with the networking infrastructure that integrates the software elements and overall program administration.

Significant technical progress toward a system enabling virtual environments for training has been achieved during the second quarter of the effort. We have arrived at an initial usage model for the completed Training Studio software that has helped guide our design and development. We have developed system capabilities along the lines of our proposed system architecture, and now have the primary software components, Vista, V-RIDES, and STEVE (Soar Pedagogical Agent) plugged into the communications bus system, exchanging information using the TScript message protocol and effecting instruction in a virtual environment for initial tutor examples.

To further guide our system development, we have identified and started development of a Navy-relevant application domain, the operations and maintenance training of the high-pressure air compressor (HPAC), which serves as a stressing case for visual, human modeling, and simulation complexity. As a part of the larger ship-board Gas Turbine system, our work on the HPAC domain will contribute to developing a training application common to the majority of newer Navy vessels.

Methods, Assumptions & Procedures

This section discusses the premises upon which we are conducting system development. These premises include a usage model, a software design for distributed update, and aspects of the initial domain upon which the system is tested.

Usage Model

The usage model for the completed Training Studio system is important for several reasons. At this stage it is important in guiding our development efforts to support the planned usage. Soon it will be important as a skeleton structure for user guide material explaining how to use the system. This is an initial user model, and feedback or insights on refining it are most welcome.

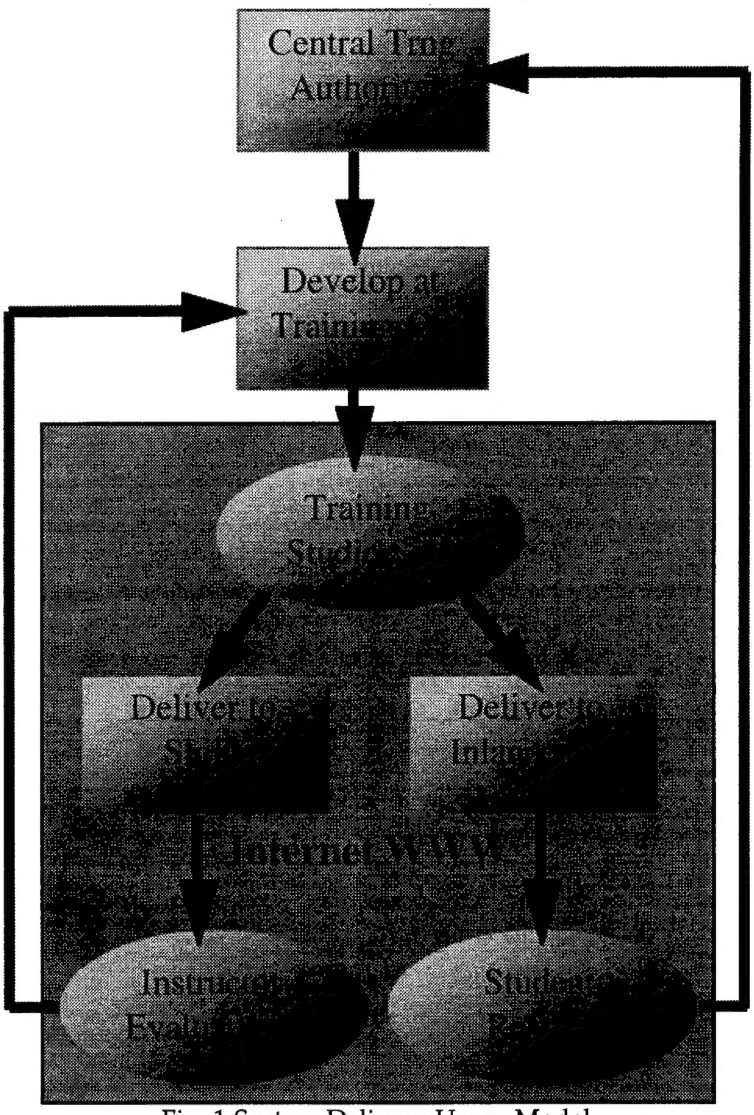


Fig. 1 System Delivery Usage Model

We started with an idea of how training material is to be delivered and used as part of Navy operations over the Internet or a similar closed network system. The delivery approach can be adapted to other services or organizations which are dispersed.

In the delivery usage model (see Fig.. 1), a demand for training in a particular topic comes from a central training authority which is tasked with prioritizing and funding training efforts. This request is handled at a Training Center which has the appropriate hardware, software, and personnel to develop the training material. More detail is provided on the development model later. A course using the Training Studio is developed, and fielded to inland reserve areas and out to ships at sea using the World Wide Web protocols on the Internet or a similar but closed (secure) network system. The ships and inland reserve sites involved should have equipment sufficient to run the Training Studio software, and one or more people who are responsible for updating/maintaining the software and hardware.

After delivery of the Training Studio simulations, students use the system to train with, and people at the site with experience in the suubject domain evaluate the courses and simulation for effectiveness (these may be actual instructors or specialists). The results associated with each student in the course are relayed to the central training authority over the same network, and the results of instructor evaluations are returned to the Training Center for use in refining the system (the student evaluation results could be used there too). Unlike centers using traditional curricula, the training center described here could acton on the requested changes and update the sites using the curricula fairly quickly. Of course, this presumes that beureaucratic approval is kept to a reasonable level for the changes, but in periods of war and other extreme competition, beuracracy is often kept to a minimum.

We are taking steps in the system development to support distributing the training simulations using the World Wide Web since given the large amount of data for 3D models and simulations, it is somewhat of a technical challenge, but have not at this point built support for relaying instructor evaluations of the system or the student scores in training back out of the system. This feedback could be developed fairly easily using current WWW resources.

The actual development of a course for a given domain starts at a Training Center with defining a Course Structure (see Fig. 2). Here the developers get an idea of what material needs to be covered, what simulations may be helpful, what the tasks in the domain may be, and which 3D models will be needed. This gives the developers a shopping list and an initial course of action.

Initial training simulations are developed, most likely in 2D using RIDES or a software with similar capabilities, and at the same time members of the development group assess existing D model resources and obtain these. Soon after initial simulations are developed, they are augmented to updated the 3D scenes in the Training Studio using TScript messages. At each stage, the simulation is evaluated by the development group to see if it matches real system behavior in the areas that count for training transfer. When the simulation is detailed enough to support a given task in a 3D setting, the instructional developers use the Training Studio software to provide task examples. Simpler examples of tasks can be stored using RIDES, and more complex task examples where the tutoring system may be asked to recognize task variants or provide explanation of task steps to the student will be captured by the STEVE software.

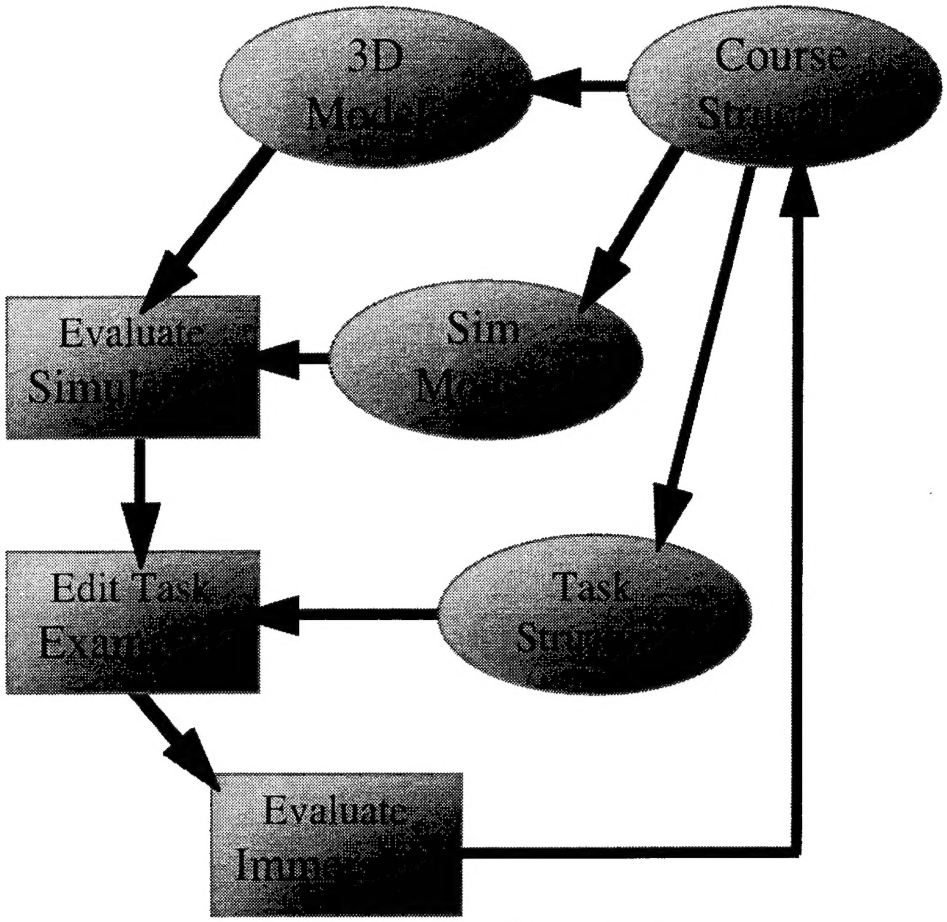


Fig 2 Development Usage Model

Several types of simulation models must be built during domain development. In our usage model, a good portion of support for capturing the simulation models themselves is present in RIDES, and we are building support in the STEVE software for capturing task models related to the student. The 3D model data used in the virtual environment is one area which can be labor intensive, and where existing data may be available to re-use. So in our usage model we have further elaborated on how we see 3D model data entering into system usage (see Fig. 33).

The usage model for using 3D models recognizes three forms of models: engineering (CAD) models and already existing models which can be purchased, the set of necessary 3D models not already existing, and those 3D models which because of the nature of the domain or its explanation, must be dynamic (mutable while using Vista).

For existing engineering models, it is often the case that the polygon count is too large for real-time interaction in a virtual environment. This is often the case because the engineers that save the model files to their tessellated (polygonal) form for rendering or transfer do not tessellate the models for lower polygon counts. The ideal place in the process to reduce polygon counts for engineering models is when using the CAD modeler involved in building the models. Often the models can be re-tessellated to a much lower polygon count using the CAD data and the original modeler.

However, most often we are not that lucky. Polygons in complex engineering models can be reduced using decimation, whereby parameters are set such as length of the polygons or area relative to their importance during decimation, and vertices for less important polygons are

removed to create fewer polygons. One of the easier-to-use commercial packages for decimation is strangely enough available from Silicon Graphics in their WebSpace Author software, a tool for generating VRML scenes. This is what we use in our domain development. Research in other advanced methods for decimation have been funded by the Office of Naval Research, and could fill this place in the process [Renze-Oliver 96].

Any necessary models not found existing should be built using a modeling system. At times a project will have to resort to generating their own special 3D models. We have found MultiGen Flight and the SGI Inventor toolset to be useful in this regard, and soon will be using the EZ3D modeler. There are many commercial modeling systems available, and for our purposes, if their output can be translated to VRML , they can be useful. During the second quarter, we worked with SGI to finish a Performer utility that saves to the Inventor 3D model format, allowing the wide range of 3D model formats that Performer loads to be translated to Inventor and from there to VRML 1.0. This was done in support of the "translate to VRML"step in the usage model. Some notable file formats that Performer loads and can now save as Inventor include: the S1000k DoD terrain format, Autocad DXF and 3D Studio, Wavefront, Lightscape, Multigen Flight, Coryphaeus Designer Workbench, and many others.

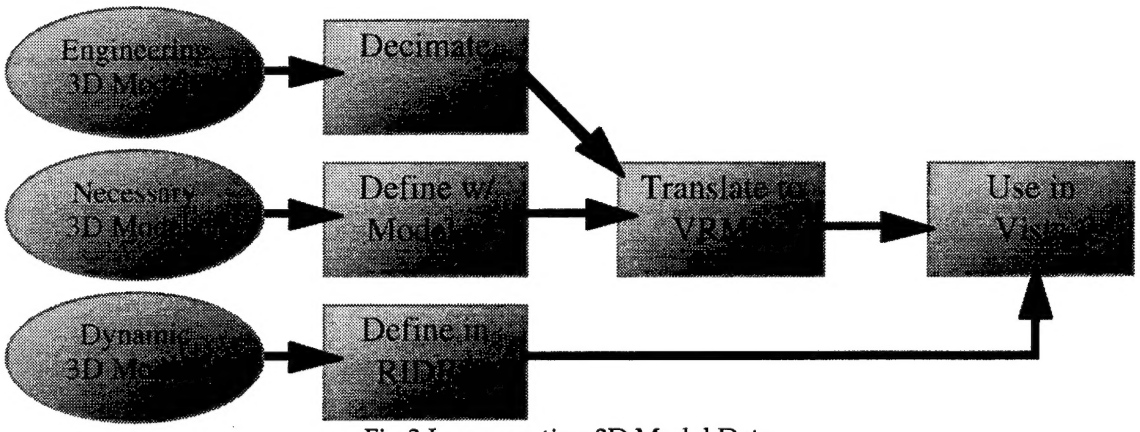


Fig 3 Incorporating 3D Model Data

The third category of 3D models, dynamic models, are those whose geometry must change live during the simulation usage. This occurs often during explanation, where for example a polygon representing the arc of an angle must change to represent a new angle value, or a visual line showing a link between two objects must change when the objects move, etc. These models are best defined using the graphical primitives in Vista, from the simulation where their values are calculated (such as RIDES).

Software Design

The software design section covers assumptions and methods used to structure the software development efforts. Some aspects of the software design choices also appear in the results section.

Communications

During the second quarter of the contract we extended the design of communications between Soar, V-RIDES, and Vista. The major thrust was in providing for communications to and from the Soar agent, and enhancements to the communication functionality of V-RIDES.

Commish

Much effort was spent on the design of a method to allow the Soar Agent (STEVE--Soar Training Expert for Virtual Environments) to interact with V-RIDES and Vista in order to demonstrate parts of the HPAC pre-start procedure. This was accomplished by designing a Tk shell (called "Commish") that could act as an intermediary between STEVE and the Vista communication bus. Commish takes high-level action commands from STEVE and sends appropriate TScript messages to V-RIDES and Vista via the communication bus. Similarly, Commish monitors the communication bus for messages from V-RIDES and Vista. This separation of STEVE and Commish provides increased modularity, allowing STEVE to operate at a cognitive level without getting bogged down in the low-level details of communication with V-RIDES and Vista.

V-Rides

A crucial technical requirement of the VET effort is the incorporation into a descendent of the RIDES program, called V-RIDES, of capabilities for communicating with Vista Viewer and with Soar. V-RIDES must be able to communicate attribute values of interest to the Soar and Vista programs. It must also be able to 'hear' and respond to relevant messages from Soar and Vista. Extensions to the communications functionality of V-RIDES was made in increments, resulting in three major releases during the second Quarter. Details on each release is provided in the Software Development/V-RIDES section of this document.

Soar Planning Architecture

At the core of STEVE's demonstration capabilities lies a general architecture for constructing and executing plans. During the fourth quarter of 1995, Rickel reviewed the planning literature in search of a framework that would allow flexible interleaving of plan construction, execution, and revision, and he settled on the IPEM architecture (described briefly in the last quarterly report). During the first quarter of 1996, Rickel implemented key parts of this architecture in Soar, as described in the Software Development/Planning and Action Execution section of this document.

Soar Authoring

To make STEVE into a practical tool for course authors, it must be possible to teach STEVE the procedures of a new domain without resorting to Soar programming. During the first quarter of 1996, we began approaching this issue in two ways. First, we are developing a simple declarative language for specifying procedures. This language allows a course author to specify the steps in a hierarchical plan and the ordering dependencies among the steps. To help the author describe the individual steps of a plan, we are developing a library of generic actions, such as pressing a button or reading a gauge, which can be instantiated to describe actions in a new domain. Our goal is a declarative language that allows course authors to describe a new domain at a relatively high level; the appropriate Soar productions that allow STEVE to operate in that domain will be built automatically from the high-level description.

As an alternative approach to the authoring problem, we have been working with a graduate student, Rich Angros, on an approach based on programming by demonstration. In this approach, the author would demonstrate new domain procedures to STEVE by actually performing them in the virtual world. This is a more ambitious approach than the use of a declarative language, and our work to date has been very preliminary. Angros has been working primarily in two areas so far. First, he has made considerable progress in formulating the problem and understanding the issues. Second, he has begun studying a program called

Instructo-Soar, developed at the University of Michigan, which we believe will provide the best starting point for designing a solution.

We have followed several plans for work in the second quarter of 1996. First, we plan to continue developing STEVE's knowledge of the engine maintenance domain; as a start, we plan to work toward a version of STEVE that can demonstrate the entire pre-start procedure. Second, we plan to improve STEVE's ability to generate explanations of his decisions and actions; by coupling Debrief with STEVE's current demonstration abilities, we plan to work toward a version of STEVE that can be interrupted at any time during a demonstration so the student can ask for more details on STEVE's rationale. Third, we plan to design and implement a student monitoring capability; specifically, we plan to work toward a version of STEVE that can monitor a student performing a task and provide help at any step upon request. Fourth, we plan to incorporate a Jack human figure into STEVE's demonstrations.

Initially, our goal is to allow STEVE to use fixed Jack poses to illustrate spatial manipulations in the virtual world. We have made some progress towards that goal, but more work remains. Finally, we plan to continue working on authoring capabilities for STEVE, including extending my declarative language and library of generic actions as well as working with Rich Angros towards programming by demonstration abilities.

Domain Selection

This section presents the Lockheed-Martin VET development team's recommendation for using the High Pressure Air Compressor (HPAC) domain for demonstration and development of the Training Studio. It is expected that the HPAC domain can be productively used as a component of the larger ship-board Gas Turbine Engine domain.

The domain selection process for the Virtual Environments for Training system was completed on February 2, 1996. The team's recommendation to the ONR was to use the High Pressure Air Compressor (HPAC) for the Gas Turbine Engine as domain for demonstration and development of the Training Studio.

Preliminary research for domain selection began during the first quarter of the contract as is documented in the VET Quarterly Report dated January 1996. At that time, Engine Maintenance was tentatively chosen as the lead candidate for the demonstration domain. As a result, more effort was spent during the second quarter researching the Engine Maintenance domain, including a trip to the Great Lakes Training Center by two members of the development team on January 24, 1996. Representatives from BTL and Lockheed Martin visited the multi-story Gas Turbine Engine simulators and discussed a number of possible domains with Great Lakes Training personnel before choosing the HPAC for recommendation to the VET team.

A second group from USC/ISI then visited the NPRDC, namely Mike Cowen, Barbara Morris, and David Dickason, in San Diego. They reviewed their materials in the propulsion domain, and discussed simulation requirements for that domain. The VET development team met on February 2 to discuss the results of each trip and agreed to recommend the HPAC domain to the ONR for final selection. This recommendation was, in turn, submitted to the ONR and we are awaiting final decision.

Overall, the domain selection and knowledge acquisition process has gone quite smoothly. We appreciate the efforts of individuals at ONR, NPRDC, Great Lakes Training Center, and elsewhere in supporting this activity.

Domain Scenario Demonstration

BTL work on the HPAC domain involves four phases, which will be cycled through several times in an incremental process of tutor design and development. They are:

- Domain knowledge engineering
- Integration with Vista models of the domain
- Development of tutorial segments in V-RIDES
- Integration with Soar

The Lockheed Martin groups efforts in domain development consist of the following:

- Acquiring and publishing on the Web photos and videotapes of the HPAC domain for use by the other development groups in the VET program.
- Generation of 3D models for HPAC control systems based on photos and videotape data
- Contacting and working with San Diego Naval Training Center to secure 3D engineering models for the Arleigh Burke engine rooms where the HPAC is situated.

The ISI efforts in the HPAC domain consist of:

- Visiting the San Diego Naval Training Center
- Arriving at an initial HPAC task structure for the STEVE agent to perform

Knowledge Engineering

Domain knowledge engineering is being carried out by building interactive models of the HPAC control panels and subsystems. In many cases, purely 2-D models will have been built first. The V-RIDES approach requires that a behavioral model of the device or complex system that is the subject matter of the tutor must be created that provides a level of simulation accuracy that is appropriately detailed and accurate for the envisioned target tutorials.

Vista/V-RIDES Model Integration

Integration with Vista models of the domain involves extending the 2-D models so that the behaving objects that were authored during the domain engineering process can create 3-D instances of themselves in the Vista environment. When feasible, objects are designed to be generic and editable, so that they can be duplicated, modified, and reused as appropriate.

V-RIDES Tutorial Development

Development of tutorial segments is carried out by using the V-RIDES authoring tools to build brief instructional vignettes. The initial instructional authoring may be carried out in the 2-D graphics world that was built in the domain engineering process (see Fig. 4 below). The instructional items are then modified, using the V-RIDES editors, for appropriateness in the virtual environment. Completed vignettes are tested in the Vista world and modified as necessary.

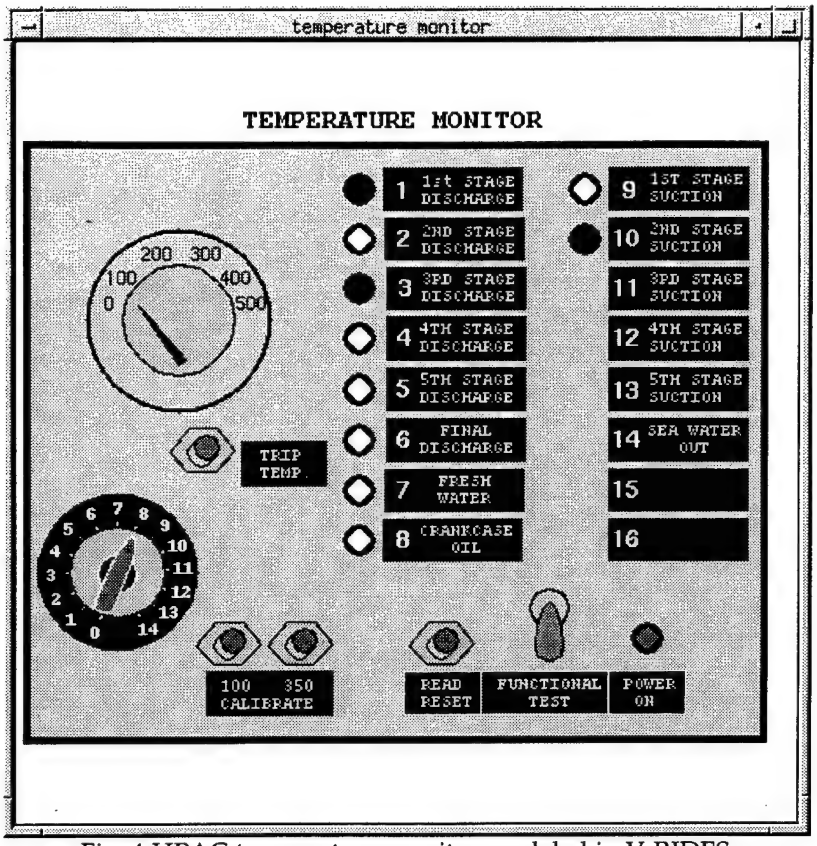


Fig. 4 HPAC temperature monitor modeled in V-RIDES

Soar/V-RIDES Integration

Soar advisor expertise can only be provided if Soar is aware of student actions and observations. In order to facilitate the authoring of many procedural training segments, the V-RIDES author marks certain simulation attributes as *control* attributes—ones that are directly set by student actions, or as *indicator* attributes—ones that hold values that are displayed to students in the simulation. Soar can register an interest in these attributes, so that the Soar expert always knows what the student is doing and observing.

Results and Discussion

This section of the quarterly report relates what we have accomplished so far, and relates the significance of these efforts to the proposed goal of effective training using virtual environment technology.

Software Development

Significant technical progress toward a system enabling virtual environments for training has been achieved during the second quarter of the effort. We have developed initial system capabilities along the lines of our proposed system architecture, and now have the primary software components, Vista, V-RIDES, and STEVE (Soar Pedagogical Agent) plugged into the communications bus system, exchanging information using the TScript message protocol and effecting instruction in a virtual environment for initial tutor examples.

Vista Virtual Environment Development

The virtual environment system associated with each student participant in the Training Studio is called Vista. Vista acts much like an X-server for one participant, updating the visual scene in response to TScript messages, and outputting messages representing the student participant's actions while immersed. This flexible approach allows us to provide general functionality which can then be used by any program communicating with Vista. General functionality for dynamic text display, movie display, spatial queries, editing, manipulating, and saving 3D scene data, and Virtual Reality Modeling Language compatibility has been developed within Vista for the VET effort.

Activity on the Vista Viewer was concerned primarily with porting Vista to Performer 2.0 and integrating the Movie and Text classes. In our previous version of the Vista Viewer we were unable to resolve relative URL's for VRML downloads from Netscape. This problem has been fixed in the current version.

As part of our early development efforts, we adapted Vista to use the Performer 2.0 software. This upgrade from Performer 1.2 resulted in improved rendering, file loading of a larger set of 3D file formats, and interaction capability, provided enhanced analytic geometry capability for polytopes we will use for advanced scene culling, and fixed bugs we had previously lived with in terms of geometry intersection and geometric analysis.

Working with developers at Silicon Graphics, we developed and tested a dynamic library for use with any Performer-based system that allows saving elements of the Performer scene graph to file in the SGI Inventor file format. We then made the source code for saving to Inventor files available to the rest of the Performer development community world-wide, where there had been many requests for this capability. This file-saving capability was incorporated into the Performer-based Vista software in support of instructional authoring capability. A scene graph can now be modified using Vista and then saved to file for reloading during instruction.

Our own network protocol, TScript, has been enhanced to support communications demands from the Soar and V-RIDES systems. Examples of new message capabilities are support for queries of named portions of the scene graph's bounding boxes and bound sphere regions, which include world coordinates. These and other messages allow different processes on the communications bus to find out the state of the virtual environment for a given student participant.

After we decided upon the HPAC training domain, we began developing initial 3D models for HPAC test domain in the VRML file format, starting with controls and monitors, the more

complex systems for student interaction (see Fig. 5 below). A part of our usage model for the completed Training Studio system depends on the Internet distribution of virtual environment training materials, and VRML is the appropriate format for this distribution, as there are a number of modelers and browsers besides our Vista viewer that can use VRML format files. Another part of our Training Studio usage model is support for re-using 3D models developed for engineering or other purposes. To effectively realize re-use of 3D models, we found that it is necessary to provide hooks that other system components such as V-RIDES can use to manipulate them in the course of training simulation. We modified the VRML file loader for Vista to support user-defined labels in the VRML files that map through as named nodes in the Vista scene graph once they are loaded into the virtual environment. In this way, changes to object position, orientation, visibility, etc. can be effected using TScript messages which reference these named scene graph nodes.

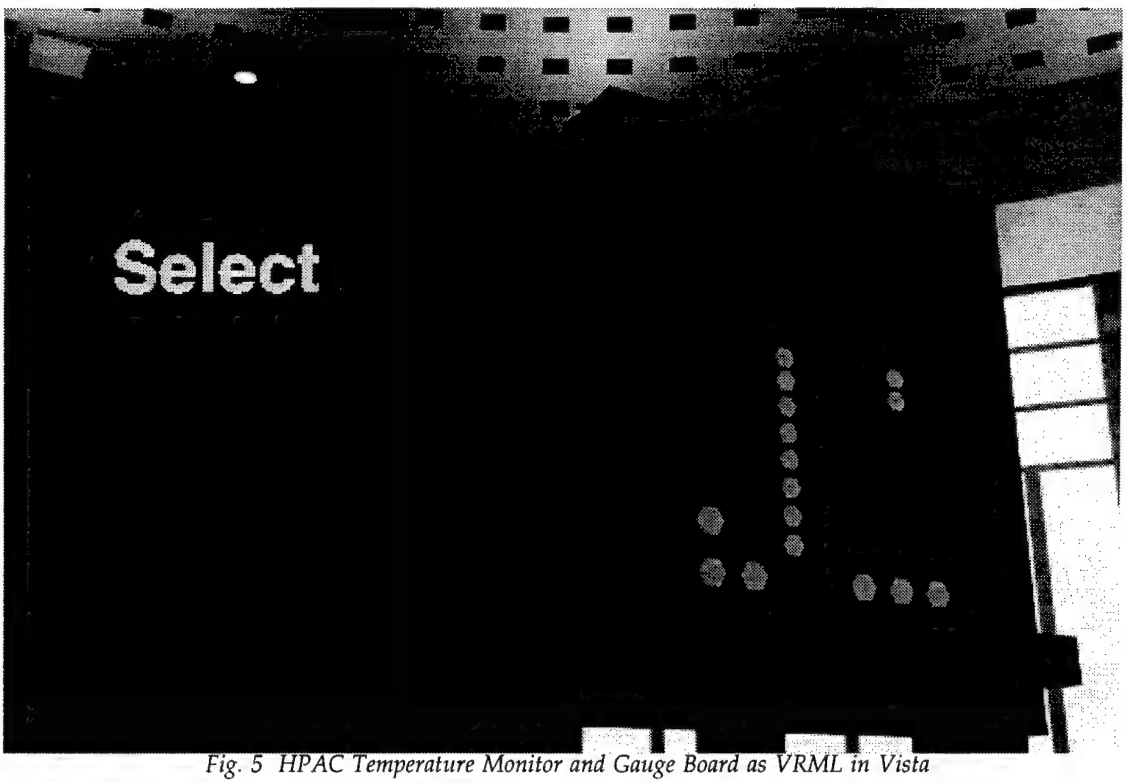


Fig. 5 HPAC Temperature Monitor and Gauge Board as VRML in Vista

Movie Playing Capability

Video clips can form an important part of a training program and therefore it was considered important that movie playing capabilities be added to the Vista viewer and also to allow movie files to be downloaded from the Internet and played within Vista.

The idea behind adding the movie playing capability within Vista was to provide a means to display already existing video clips of training scenarios (for e.g., a real-life video clip of fires on ships and how to stop such fires) in an immersed environment. This would give the instructor a state-of-the-art tool for setting up the training scenarios. The screens on which movies are to be played could be attached to already existing scene geometry and played ,for e.g., on a wall of a room inside a VRML ship model.

The Movie classes which were developed in a standalone mode on Performer 2.0 were integrated with the new version of Vista. Memory related bugs had to be fixed. At this time we are able to play small movie files within Vista. We are working towards being able to play larger movie files. TScript messages for playing movies interactively are being set up

Text Creation Capability

Vista has long had the ability to display text messages overlaid onto the scene, so that they are known to be visible to the student. Through experience with the ground controller training domain for the Air Force, and initial experience in the HPAC domain, the need for dynamic text display in the scene was evident. We have completed development of enhanced in-scene text display based on underlying Performer capability, and made this available through TScript messages. This text can be flat 2D text, extruded 3D text, or generated texture text, with arbitrary coloring and fonts. This text capability is very important for generating equipment labels, sending prompts to the student, and displaying dynamic values such as those associated with simulation. It is not sufficient to assume this information can be conveyed to the student by audio means such as spoken text, since more than one changing value may need to be displayed at the same time to match the tasks in a training domain.

A text class was developed which combines the Performer font, string and text classes, with the goal of making text creation and manipulation easier. This class was developed in a standalone mode and was later integrated into the Vista framework. TScript messages have been set up for creating 2D, 3D, and textured text. The message string and the text attributes can be changed on the fly through TScript messages. The text class does not run in the single process mode and we are working towards resolving this problem. However, it works in a multiprocessing environment. This class will make it easier for users to manipulate text and thereby make it easier for them to read textual information while immersed.

Virtual Reality Modeling Language

We have finished development of the VRML 1.0 (Virtual Reality Modeling Language) file loader for Vista, making it one of the few systems supporting immerse VRML in existence. The VRML loading capability for Vista allows VRML 1.0 files to be downloaded from the World Wide Web into Vista scenes using the HTTP protocol. VRML is a language for describing multi-participant interactive simulations - virtual worlds linked via the Internet. VRML is the standard for 3D graphics on the Internet. The fact that many existing models are in the VRML format and that many translators to the VRML format exist means our VRML loader enables existing models to be downloaded and used for immersive training, furthering reuse and rapid exchange, and reducing cost and time of development.

The previous version of Vista would crash while loading VRML documents with a large number of Inline and Anchor Nodes. This problem was related to a memory bug which has now been fixed. In the current version it is possible to load VRML documents with a large number of Inline and Anchor Nodes through Netscape or via the TScript messages.

In the previous version of Vista we were unable to resolve relative WWW Uniform Resource Locators (URLs) for VRML files downloaded from Netscape. Relative URLs are web "addresses" that are relative to the site where the main VRML file (or other file type) originated. This has been fixed in the current version of Vista. Now in order to resolve relative URL's. Vista correctly uses the full URL provided by Netscape and uses this to resolve relative URL's.

We have built and tested support for VRML inlined texture files within VRML scene files which allows texture files to be downloaded from the Internet and displayed within the Vista framework. This work has been done in a standalone mode and will be integrated in a future release of Vista.

Viewing Frustum Events

The code in Vista for defining arbitrary frustums has been developed and tested. In addition to the user's frustum there can be other viewing frustums which correspond to SOAR agents' views. These frustums are referred to as arbitrary frustums, arbitrary viewing frustums were defined and aligned with the user's frustum for testing. Certain objects in the scene were tagged as of interest to a arbitrary frustum. When these objects of interest came within the scope of a arbitrary viewing frustum or went out, an event would get triggered which would tell us if the object of interest was visible or not visible. Some bugs in the code were fixed and now the arbitrary frustums events work correctly in Vista.

For detailed information on using Tscript to create movies, text, and events, see *Appendix A: VET Communications Bus.*.

V-RIDES Development

During this period, BTL released three major versions of V-RIDES to ISI and Lockheed Martin during this quarter, providing successively enhanced communications functionality.

On February 1, V-RIDES 4.2.1 was released. This version supported the sending of TScript messages from a V-RIDES simulation. It also supported the linking of V-RIDES attributes to particular TScript message types. This makes it possible for a V-RIDES tutor to deal automatically with each type of TScript message of interest.

On February 29, V-RIDES 4.2.3 was released. This version supports attribute serving. This feature lets Soar or Vista express an interest in a particular V-RIDES attribute. Thereafter, whenever that attribute changes in V-RIDES, its new value is exported to the collaborating application. This feature is of particular interest to Soar, which must be able to monitor the student users' manipulations of simulated controls.

On March 19, V-RIDES 4.2.4 was released. This version improved attribute serving features and eliminated a keyboard mapping bug that made it difficult to use V-RIDES from an X-server on an SGI workstation.

For detailed information on the communications facilities supported in V-RIDES, see *Appendix B: VET Communications in V-RIDES.*.

Soar Agent Development

ISI's primary accomplishment during the second quarter of the VET effort was the design and implementation of a Soar agent (STEVE) that can interact with V-RIDES and Vista in order to demonstrate parts of the HPAC pre-start procedure. STEVE can walk the user through several steps of the pre-start procedure, including pulling out the dipstick and checking the oil, checking the power light on the condensate drain monitor, pressing the function test button and checking that all alarm lights come on, and extinguishing the alarm lights by pressing the reset button. For each step, STEVE gives a spoken explanation (using the TrueTalk text-to-speech program), points to relevant graphic object using a disembodied hand, performs actions in the virtual world by sending messages to Vista and V-RIDES, and monitors the virtual world for the effects of the actions.

STEVE Communication Support

Commish, the Tk shell designed to facilitate interaction between STEVE and the Vista communications bus, was implemented to provide STEVE efficient access to the state of the simulated world. V-RIDES controls the simulation and stores the simulation state in attribute-value pairs. To provide STEVE access to these attributes, ISI and BTL personnel collaborated to design a TScript protocol by which V-RIDES can broadcast attribute changes to the communication bus. Commish monitors these messages and maintains a copy of the current state of the simulation. On each decision cycle, STEVE asks Commish for an update of the current state and for any important events that occurred since the last update. Thus, Commish spends most of its time maintaining a model of the simulated world, freeing STEVE to spend more of his time making higher-level cognitive decisions.

Planning and Action Execution

During this second quarter, STEVE's general architecture for constructing and executing plans was developed. The framework of this architecture is based on the Integrating Planning, Execution and Monitoring (IPEM) architecture. In order to demonstrate a task, STEVE constructs and executes a plan for the task. Currently, STEVE constructs a plan by repeatedly decomposing high-level steps into more primitive ones, resulting in a hierarchical plan. Although STEVE currently fully constructs a plan before executing it, the planning framework has been designed and implemented to allow more sophisticated planning abilities in the future, such as flexible interleaving of plan construction, execution, and revision. We expect STEVE's use of an explicit plan to allow him to explain his actions to students as well as evaluate student actions, and we expect the flexibility of the planning framework to help STEVE cope with unanticipated events in the virtual world and unanticipated student actions.

Debrief

As described so far, STEVE can demonstrate tasks and explain each step along the way, but STEVE cannot answer student questions. To allow the student to ask why STEVE performed an action, and why other actions were not performed instead, we spent considerable effort this quarter integrating Lewis Johnson's Debrief program with STEVE. Debrief is a large Soar program (over 1700 productions) that was developed as part of a separate project to allow Soar fighter pilots to explain their actions and conclusions.

Integrating Debrief and STEVE involved three main activities. First, we converted Debrief to the current version of Soar. (Debrief was originally written for an older version.) Second, we teased out its dependencies on the fighter pilot domain, resulting in a version of Debrief that is more domain independent. Finally, we provided the knowledge Debrief needs in order to operate with the knowledge representation used by STEVE. Although we have not yet tested all of Debrief's functionality, we successfully integrated Debrief with STEVE's demonstration of the pre-start procedure, so we can now begin using Debrief to answer student questions.

Demonstration Scenario Development

Although the final selection of the domain has not been confirmed by the ONR, we have continued work in the HPAC domain based upon discussions with the VET Program Officer. While at the Great Lakes Naval Training Center, photographs were taken of the HPAC in one of the Gas Turbine Engine decks. In addition, the instructors at Great Lakes videotaped the start procedure and forwarded to us. We have since digitized the photographs and are in the process of transferring the video to mpeg format. Mike Cowen, NPRDC, has also been extremely helpful in sharing his knowledge and resources from a similar project. He is

currently having existing CAD models of the Arleigh Burke translated into a format that we will be able to use on VET. He has almost completed one zone, which we are in the process of arranging to download the model.

Knowledge Engineering

A very substantial portion of the behavior of interest of the HPAC system has now been modeled in V-RIDES. This knowledge engineering process consists of building simple interactive 2-D graphic models that capture the relevant behavior. A set of 2-D graphic scenes with interactive behavior of a significant subset of the HPAC controls and indicators has been developed. The HPAC panels and subsystems that have been simulated to date include the following:

- Condensate Drain Monitor Panel
- Temperature Monitor Panel
- AC Magnetic Motor Controller Panel
- Gauge Board

In addition, a number of ancillary simulation modules—for use in simulation and tutorial authoring—have also been developed. These include:

- Temperature Setting Controls
- Pressure Setting Controls

Vista/V-RIDES Model Integration

The second phase, integration with Vista models of the domain is now underway. This phase of development requires that the objects of the V-RIDES simulation be given additional capabilities so that they will be able to create analog manipulatible objects in the Vista world. A preliminary methodology has been developed for adding this functionality to a V-RIDES object. Each type of object is given an object event called Init that is responsible for creating its analog in Vista. Other standard object events are ChangeColor, which is responsible for controlling the color of the Vista objects, and respond_to_pick, which handles simulation responses to a selection event on the control's analog object in Vista. For wide types of V-RIDES objects it is not necessary to edit these events. They make use of local data stored in the object's attributes. When an author duplicates such an existing object, he or she modifies the attributes that control the features of the object that it creates in Vista.

This phase is now well-advanced. Good renditions of the control and indicator objects on the Condensate Drain Monitor, Temperature monitor, ACMagnetic Motor Controller, and Gauge Board panels are now generated in the Vista HPAC model environment. The controls on these panels respond to user manipulations, whether they are carried out in Vista or in the 2-D 'scene windows' that represent those panels.

Tutorial Development

An approach to authoring tutorials in V-RIDES is under development. One technique that shows promise is to rough out a simple tutorial exercise—such as one that teaches the approved sequence of operations for a procedure—by using the patterned exercise authoring tool in the 2-D simulation windows. Then the V-RIDES custom authoring tool is used to edit some of the component instructional steps in the tutorial so that they will work appropriately in the virtual environment.

This phase of the work is still preliminary, but a small tutorial that teaches the student how to perform the functional check of the condensate drain monitor has been developed.

Soar/V-RIDES Integration

The low-level V-RIDES functionality that supports registration of Soar interest in V-RIDES control and indicator attributes has been implemented in its first version. Efforts are now underway to test the functionality by having Soar observe and remediate student actions during procedures.

Contract Administration and Resources

This section covers the administrative efforts associated with the three development groups in the VET effort, ranging from finalizing sub-contracts to securing hardware and software for the program.

Subcontracts

The subcontract between Lockheed Martin and BTL was finalized in early March. The subcontract between Lockheed Martin and USC / ISI was finalized and executed in late February. Although the finalized subcontracts were delayed from the start of the prime contract, it did not seriously impede our ability to perform work toward the contract, since Lockheed Martin was able to forward fund the subcontracting groups. The only consequence of the delay was that ISI was not able to purchase our complete Flock of Birds tracking system as quickly as we anticipated. The reason for this is detailed in the next section under Hardware and Software Resources/ISI Resources.

Hardware and Software Resources

Some contract effort has been devoted to obtaining hardware and software resources, and the results of those efforts are reported here.

Lockheed Martin Resources

All hardware reported in the first VET Quarterly Report dated January 1996 as approved for purchase from Lockheed Martin Corporate funds has been received and installed. Equipment now available at the Lockheed Martin Advanced Technology Center for primary VET development use includes:

5 Silicon Graphics Workstations:
- 1 Onyx Infinite Reality Engine with 2 R10000 CPUs, 1 RM6 graphics board
- 2 WebForce Indigo High Impact Workstations
- 2 Indy XZ Workstations
- 1 Virtual Research System VR4 Head Mount Display
- 1 Ascension Technology Flock of Birds Extended Range Position Sensor Assembly

Development software totaling $30,000 for the above machines, purchased using Lockheed Martin software funds, were also received and installed during this period.

BTL Resources

Crucial items of hardware and software were acquired in second quarter of the VET effort.

- **Hardware acquisition efforts.** In coordination with Lockheed-Martin, Forms 1419 were completed to obtain authorization to purchase the requisite SGI workstations for BTL. Once USC received authorization and budget, the systems were ordered immediately. On February 14, they were delivered to BTL. They have since been integrated into the lab's local network, and they have been intensively utilized during the past two months.

- **Software acquisition.** A license for the SGI Varsity Pack software package has been purchased from USC's University Computing Services. BTL has also purchased and received the ToolTalk Developer's kit and Performer. These software packages will be used in the development of V-RIDES for SGI workstations.

- **Network access.** Under funding from an Air Force contract, a T1 line has been installed from USC's University Computing Services to BTL. Additional hardware and software integration has been carried out by USC's University Computing Services and by the BTL staff. Efforts are underway to provide FTP site access and web page documentation for our research collaborators.

ISI Resources

USC / ISI issued purchase orders for two Silicon Graphics workstations (a High Impact and a Maximum Impact), a Virtual Research VR4 head-mounted display, Performer software, TrueTalk speech generation software, and for DIS and stealth packages from MaK Technologies. Most of this equipment has arrived, with exceptions noted below. ISI has purchased a share of a file server on which to store VET project data. USC/ISI already had a license for Jack on one workstation. The Jack group agreed to provide us with additional Jack licenses free of charge for the duration of the VET contract, for use on the two new SGI workstations. We are grateful to ONR for their efforts on our behalf in this regard.

After receiving the Maximum Impact, ISI discovered that it is missing a full complement of texture memory, as well as the ICO board for driving a head-mounted display. This came as an unwelcome surprise, and only now have we received updated delivery dates for these items. The texture memory is scheduled to ship in May, and the ICO board in June. The lack of an ICO board means that we will not be able to run the head-mounted display in stereo mode right away; we will still be able to work immersed with single displays for the time being.

The delay in finalizing the subcontract with Lockheed Martin affected the purchase of the Flock of Birds. Our original equipment budget assumed that USC would liable for sales tax on all equipment. Once we received details of the contract provisions, we found that this was not the case, and that we would therefore be able to stretch the equipment dollars further. In particular, it allowed us to move up our purchase of an extended range transmitter for the Flock of Birds into the first year of the contract. This is a net savings to the project, since our initial plan required us to purchase a standard range transmitter first, and then discard it when the extended range transmitter was purchased. Unfortunately, the cost of the extended range transmitter was beyond the scope of what was allocated under forward funding, so we could not actually issue the purchase order until the contract was signed. The Flock of Birds has just arrived at ISI, we will be installing it shortly.

USC / ISI has received regular updates of V-RIDES and Vista software from USC/BTL and Lockheed Martin. Their software releases in this regard have been extremely helpful to the ISI effort.

Reviews and Integration Meetings

There have been no meetings with ONR personnel in the past three months. However, the three developing groups for the Lockheed Martin VET effort have participated in regular internal software integration meetings with other project members on a monthly basis.

Software Integration and planning meetings were held at the ISI facilities in Los Angeles on January 29 and February 28. A set of milestones testing working operation of the systems together were established, leading up to a final second quarter integration.

The final second quarter VET contract integration meeting was held at the Lockheed Martin Research and Development Division, Palo Alto, on March 31, 1996. The integration meeting involved actually running the three major software components together, discussing future directions of work and bug-fixing. We merged the latest versions of SOAR, V-RIDES and Vista in a scenario where the STEVE agent uses Vista and RIDES to demonstrate a task to the student. The preparations involved getting in touch with the Information Science Institute (ISI), USC, and ensuring that their software was running before they arrived at Palo Alto.

Conferences/Workshops

The Virtual Reality Annual International Symposium (VRAIS) was held at the Marriot, Santa Clara, CA, between April 1 and 3. Many of the people working on VET from Lockheed Martin, ISI, and BTL attended this symposium. VRAIS 96 provided an excellent forum for learning about the state of the art in virtual reality hardware and software, to hear about ongoing research in a number of directions, and to talk with the other people working in this area. At that meeting we met with other personnel connected with the VET program, e.g., Carol Horwitz and Bowen Loftin.

As a general introduction to virtual reality, almost all of the talks were useful. However, two were particularly relevant to ISI's current VET work. First, a large group from the University of Pennsylvania is working on methods to control Jack agents in virtual worlds [Trias 96]. Their work complements our work because they are focusing on lower-level control (e.g., motor skills and visual attention) than we are. Second, Billinghurst and Savage from the University of Washington are developing a program that can perform actions in the virtual world on request and can also interpret the actions of humans operating in the virtual world Billinghurst 96]. These capabilities are similar to those we are developing in STEVE, but the work complements ours because they are focusing much more on natural language dialogue than we are.

ISI has submitted an abstract to the ITS 96 workshop on simulation-based training, which was accepted. Rickel and Johnson plan to participate in this workshop, in June 1996.

Johnson's paper for ICCE '95, "Pedagogical Agents for Virtual Learning Environments", was recommended for publication in the *Journal of Educational Multimedia and Hypermedia*. We are currently extending the paper so that it may serve as a journal submission.

Conclusions

The second quarter of the VET effort resulted in significant extensions of the SOAR Agent, RIDES, and Vista Viewer which were designed to move us closer to integration of the separate systems to a final working product along the lines of the system we proposed for VET. A series of integration planning and testing sessions were conducted to validate the design in incremental development phases. This allowed problems and issues to be identified and addressed early on, as well as ensure that development progressed as effectively as possible.

Our continued participation in workshops and conferences helps to ensure that what we are doing on the VET project will be relevant to the world at large, as well as allow us to make use of current relevant findings.

References

[Ambros-Ingerson 88] Jose A. Ambros-Ingerson and Sam Steel, "Integrating Planning, Execution and Monitoring," Proceedings of AAAI-88, pp. 83-88, 1988.

[Billinghurst 96] M. Billinghurst, J. Savage, "Adding Intelligence to the Interface", Proceedings of the IEEE 96 VRAIS, March 1996, Santa Clara, CA, pp. 168-176.

[Renze-Oliver 96] K. J. Renze, J. H. Oliver, "Generalized Surface and Volume Decimation for Unstructured Tesselated Domains", Proceedings of the IEEE 96 VRAIS, March 1996, Santa Clara, CA, pp. 111-121.

[Trias 96] T.S. Trias, S. Chopra, B. D. Reich, M. B. Moore, N. I. Badler, B. L. Webber, C. W. Geib, "Decision Networks for Integrating the Behaviors of Virtual Agents and Avatars", Proceedings of the IEEE 96 VRAIS, March 1996, Santa Clara, CA, pp. 156-162.

Appendix A: VET Communications Bus

The Training Studio is a virtual environment where several people can meet for training. To support this, several Vista Viewers and agents can be assembled together on the network, and kept synchronized together using distributed protocols across a communications bus. The distributed protocols is a set of messages which change the world model of the Training Studio. Two main forms of distributed communications are supported in the Training Studio. These are our own object-oriented, training specific protocol called TScript, and the ISO standard Distributed Interactive Simulation (DIS) protocol. We developed TScript to address specific training-oriented needs in a networked virtual environment, and we adopted the widely accepted DIS protocol to open up the Training Studio to the wide array of vehicle-oriented simulations that are available for use under this standard.

The mechanism by which the processes in the system communicate information is called the communications bus, a software abstraction of a computer bus. The distributed protocol by which the processes communicate is called TScript, and the transport mechanism for this distributed protocol is ToolTalk, originally developed by Sun Microsystems. ToolTalk was chosen for the transport system because it allows processes to register interest in messages, much like a hardware component selecting certain messages addressed to it from a large set of messages on a computer bus.

Arch. Abstraction	**Element**	**Software Element**
Communications Bus	Participants	Vista(s) and Agents
	World Model	Scene Graph w/objects
	Distributed Protocol	TScript (Ontology)
	Transport Mechanism	ToolTalk
		RPC
		TCP/IP

This TScript protocol consists of groups of messages which can change visual scenes and convey changes in the distributed state of the simulation. It is not meant as a programming language, but rather as a common collection of commands (protocol) that can be used from inside of programming languages, or other tools or utilities connected to the communication bus.

The categories of TScript messages are: object control, scene graph manipulation, event monitoring, and participant control. TScript supports creating 3D objects, constraining them, modifying them, and deleting them, in one or more Vista Viewers. Information about the state of these objects is also conveyed using the TScript protocol to any other processes on the communication bus.

Since objects created by TScript are part of a scene graph, TScript supports modifying the scene graph transformations, switches, and graph structure so that objects are transformed, visible, and modifiable in a very flexible manner over the networked communications bus.

TScript supports modifying the properties of objects, such as constraints on objects. Arbitrary, named transformation matrices in Vista scene graph can be constrained for yaw, pitch and roll, as well as in the X, Y, and Z axis. The constraints can have an upper or lower limit, or both. They can also be made to preserve the same value, which is a form of object locking. Functionality for the wider range of constraints has been tested. Any independent position or orientation value for an object can be constrained. The locking functionality per

object was built in support of immersed training applications where the training methodology demands that only certain objects can be moved by the student.

We have arrived at a representative set of events that characterize a participant's interaction with the environment fairly well. TScript messages for logging and monitoring events per participant while immersed have been developed. These events are reported as TScript events associated with the participant's viewing frustum, entry of objects into spherical, boxed, or cylinder regions, as well as intersection of object geometry with line segments defining arbitrary regions. By abstracting these events, domain experts do not have to become familiar with details of computational geometry in order to find out what is occurring in a spatial setting composed of one or more uncontrolled elements, most notably the students themselves.

Using TScript messages, participant events can be registered with each Vista Viewer, and the participant's viewing and interaction modes can be controlled using TScript. TScript supports participant scope on almost all messages dealing with participants, so that individuals can be affected specifically, or a whole group can be affected by the message.

Many devices are now available for use with virtual environments, and many more are likely to be introduced. Our treatment of devices largely removes the instructional developer from having to worry about which device was used to interact or view a given instructional setting. Selection events in TScript were abstracted to the object which was selected and the location on the object, with no mention of whether it was a CyberGlove, flat-screen mouse, or Ascension 3D mouse used by the student to select an item. This abstraction applies to the viewing systems as well, the instructor does not need to be concerned about the device used to view a setting, whether it is a FakeSpace boom, head-mount display, or normal computer monitor. They can develop the course in a spatial setting knowing the student has a view onto it and a way of interacting with it, and really focus on the domain itself.

Guiding the student's view onto simulations and other 3D examples is very important in the context of training methodologies in a spatial setting. We developed several viewing modes for controlling the views of individuals in the training studio. These modes are: viewing paths with focus objects; locking the view into a sphere around a possibly moving focus object; locking the orientation of a student to focus on an object while allowing free movement; and tethering a student to an object's motion while allowing freedom of view orientation.

1.1. The Shared World Model

The shared world model is an important topic when discussing the use of the communications bus. The TScript protocol is a set of messages which modify a scene graph or a participant's interaction with a scene graph. The scene graph is a directed acyclic graph used to represent a scene. There is a root node which anchors the scene, called vrScene. As children of this node there can be transformation matrices, switch nodes, or geometry nodes.

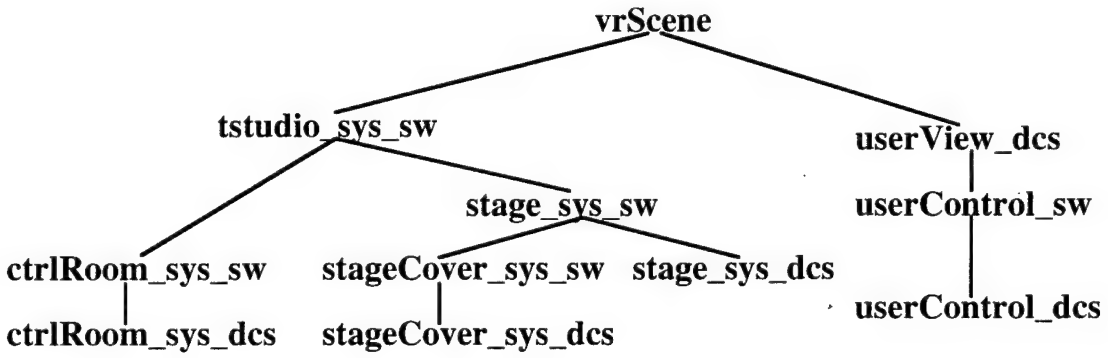
Vista traverses the scene graph from the root vrScene to the leaf nodes every time it draws a picture on the screen (actually Performer underlying does, but the scene graph is a very common concept in 3D graphics). Starting from the scene root, it traverses each child from first to last, depth first. As it traverses the scene graph it multiplies each matrix it encounters against a previous composite 4x4 matrix, and when it arrives at geometry, it uses the resulting composite matrix to multiply the vertices of the geometry. This is how you make things move in Vista, you change the values for position, orientation or scale of a transformation matrix in the scene graph, and any children below it are modified accordingly.

When Vista traverses the scene graph, it pays attention to switches as well. If it encounters a switch that is turned off, it goes no further down that branch of the tree, and so no part of that tree will be drawn. If the switch is on, it proceeds, and will eventually draw it.

Switches and transformation matrix nodes (dcs, for dynamic coordinate system) can have any number of child nodes, which may be other switch, dcs nodes, or geometry nodes. Geometry nodes may not have children.

Following is a figure representing the scene graph that Vista starts out with every time. At the root is the named node vrScene, and as one child is the training studio (tstudio_sys_sw) and as another child is the userView_dcs. The userView_dcs is updated each time Vista draws a scene to reflect the world coordinates of the user (participant) who is using vista. Interface controls that move with the user may be attached to the tree below userView_dcs, namely at the named node userControl_dcs. TScript messages to set the position etc. for userControl_dcs will move the interface object relative to the viewer, not the world.

The scene graph under tstudio_sys_sw consists of the control room and the stage, and the stage has a sub-tree representing the stage cover (roof). Objects meant for use by the instructor should always be attached to the named node ctrlRoom_sys_dcs, while object that are intended for the student participants should always be attached to the stage_sys_dcs. This is a convention in the world model that is quite useful, and should be adhered to. It is also possible to add nodes to the top-level scene itself, but it is not recommended.



Another part of the shared world model used in the Training Studio are conventions for measurement.

- Distances/Rates/Accell etc. are all in meters
- Orientation values for any angles values are always in degrees
- Colors are expressed with red, green, blue, and alpha components, and range from (0,1) as floating point numbers.
- Object priority is expressed in the range (0,1) as a floating point number.
- When using scale for objects, a scale of 1.0 will leave it at its current size. If an object is a 2x2x2 cube, and it is scaled by 10 it will have the dimensions 20x20x20, meaning it will now be 20 meters in each of dimensions.

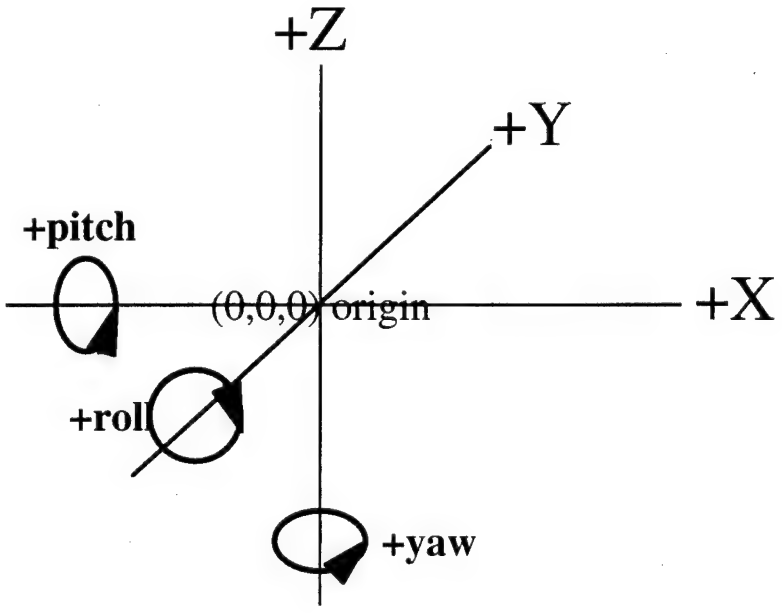
Another convention in place for the Training Studio is that it operates as a Cartesian space where angles are expressed using the right-hand rule.

When Vista is initialized, the Training Studio stage is always placed so that the middle of the floor is the origin (0,0,0) in this Cartesian space. Giving an object coordinates of 0,0,2 for

its translation means that the object will appear in the center of the floor, with the object's center 2 meters up from the floor. By convention the viewer always starts his session in Vista looking into the stage so that he can see the origin, and +X in world coordinates is to his right, +Y in world coordinates results in positions further into the screen, and +Z in world coordinates results in positions above the origin. Of course, minus values along these axis are the opposite. As the viewer moves around, the world coordinate system does not change. The same conventions apply for local coordinate systems, i.e. any and all nodes below a given transformation matrix (dcs).

Now for an explanation of the orientation conventions. There are three angles in every transformation matrix that represent an object's orientation, yaw, pitch, and roll. These are always in degrees. If your process works with angles in radians, be sure to convert them to degrees before sending them out in any TScript message.

Each angle is applied to an object in order. First the Yaw is applied to an object, then the pitch, and finally the roll. A positive yaw value is applied on the Z axis, and will turn the object counter-clockwise. A positive pitch is applied about the X axis, and will tip the front of an object upwards. A positive roll is applied about the Y axis, and will roll the object clockwise as you look down the Y axis. The front of any object is always along its Y-axis in its own local coordinate frame.



1.2. ToolTalk

ToolTalk is a TCP/IP and RPC, connection-oriented messaging system developed by Sun Microsystems and distributed by Silicon Graphics as well. Its purpose is to support messaging between software tools. In our use of ToolTalk, there is a central server (ttsession) which the components connect to. Each component registers interest in a set of messages, and when this message is sent to the server, all the clients that have registered interest receive a copy of the message.

One may think of the ttsession process as the actual incarnation of the communications bus abstraction. Each vista viewer and any other processes wishing to share the same world must connect to the same ttsession. The general user does not need to be familiar with ToolTalk, since it is just a transport mechanism, and not a separate language. They do need to know that ToolTalk is being used to check if their system has ToolTalk, and to check if the ttsession message router is up and running when troubleshooting.

1.3. TScript Protocol

The main distributed protocol used in the Training Studio framework is called TScript. The name TScript means *Training Script*. This protocol consists of groups of messages which can change visual scenes and convey changes in the distributed state of the Training Studio. It is not meant as a programming language, but rather as a common collection of commands (protocol) that can be used from inside of programming languages or utilities.

1.3.1. Object Creation

Different forms of simulation and graphic objects can be created for use and update inside of Vista. The Vista Viewer and Smalltalk framework maintain class definitions which represent most of the graphic objects. In Smalltalk, generating these TScript messages to create graphic objects in any connected Vista Viewers entails creating a new instance of a class and then initializing it to broadcast the appropriate TScript messages.

When a Graphic Object (GO) is built, Vista must attach it to part of the scene graph, and so the name of a parent node in the scene graph must be specified. When a TScript message for creating an object arrives to a Vista Viewer, it creates a switch node and a dcs node for the object, and gives them names by convention. So if you sent a vrBldGO message with stage_sys_dcs as the parent, and named the new object cube99, the stage_sys_dcs node would have a new child node called cube99_grf_sw, which by default is switched on, and this node will have a child node named cube99_grf_dcs, with the identity matrix as its default (effectively does no translation, rotation or scaling). When you add cube geometry to the graphic object named cube99 using vrAddToGO, it puts a set of colored vertices as a child of the cube99_grf_dcs node.

The convention is that all switch nodes created as part of object creation have *_grf_sw attached to their name, and all dcs node created this way have *_grf_dcs attached to their name. TScript allows the user to make single switch nodes and dcs nodes as part of the scene graph, and this can be quite useful for complex structures. By convention you should append *_sys_sw or *_sys_dcs for their names as appropriate.

1) All objects created must specify a parent
2) All arguments for TScript messages must be given
3) All objects will be created with an associated switch and dcs node

vrAddSimObj <sim-type>
Where the parameters are specific to the simulation type. Will use default model specified in any previous TScript message. There are currently only two such types of simulation object supported directly in Vista - other types, and future types should be supported in the agent framework.

satellite <name> <major> <eccentricity> <inclination> <ascension> <perigee>
<red> <green> <blue> <alpha>
Accepted by Vista Viewer to build a satellite using 5 orbital element parameters and a color specification for the path and disks. The satellite scene graph components can be referred by name as <name>_sys_sw and <name>_sys_dcs, The disk and path associated with the satellite orbit are named as <name>_disk_sw, <name>_disk_dcs, <name>_path_sw, and <name>_path_dcs.

constelem <name>
<major> <eccentricity> <inclination> <ascension> <perigee> <true-anomaly>
<red> <green> <blue> <alpha>
Similar to satellite sim type, but expects argument for true anomaly as well, so that constellations of satellites, each in correct phase with the other, can be built. Simulation should be turned off until all constellation elements of a constellation have been defined, then turned back on.

Various types of visual objects can be created and monitored in the Training Studio framework. The vrBldGO message builds a graphic object container, which you then add geometric shapes to. There is also a file-based variant which can load geometric files.

vrBldFGO <obj-name> <parent> <model-file> <X> <Y> <Z>
Build container, add file-based geometric model to it. This message is handled by Vista. The model-file can be Flight, Inventor, Designer Workbench, or other format files that Performer can load.

vrIvGO <obj-name> <parent> <Inventor2.1-ASCII-syntax-body>
Accepted by Vista (Performer 2.0 version) this message will build a graphic object specified using the SGI Inventor 2.1 syntax. One should specify a top-level separator, and then inside that, one can specify transforms, switches, LODs, geometry, etc. An obvious use is if some application connected to the communications bus needs to display 3D graphs or data live, it can send this Iventor message to have the 3D data displayed.

vrFileObj <model-file-name>
Convenience message handled by the agent framework. This will cause the Profiler to make a representation of the file object, and send a vrBldFGO to all the Vista Viewers. The file model will load into the studio stage at the point 0,0,0 with all zero orientations. It will be named <model-file-name>1_grf_dcs.

vrSceneSave <scene-name>
Accepted by the agent framework, in order to save scenes. The agent framework will iterate through all graphic objects it has a representation for, and save them to a scene file. Information on scale, translation, rotation, switch settings, color, parent, etc will be saved, and can be loaded using *vrSceneLoad*. The file saved has the extension *.scene. Since this can result in saving large scenes, which take long to load, it is recommended that scene controls in the control workspace are used to select a small set of objects to be saved in a scene.

vrSceneLoad <scene-name>
Accepted by the agent framework, this message will cause it to look for a scene file of the same name (don't add the *.scene extension) and cause that scene to be loaded in the Vista Viewer. This load will work for scenes saved using the *vrSceneSave* message, or saved using the control panel.

vrBldGO <name> <parent> <priority> <X> <Y> <Z> <yaw> <pitch> <roll> <scale>

<red> <green> <blue> <alpha>
The Vista Viewer handles this message. As a result of the message, Vista will build a graphic object container. You can then add various graphic shapes to this container.

All geometry, such as boxes, lines, etc. that are added to the graphic object container inherit its setting for color and transparency, as well as its overall coordinates.

The vrAddToGO message adds a piece of geometry to a given graphic object container. In this way, a complex, additive shape for a graphic object can be specified using the simpler geometries. The dimension parameters are always in meters.

vrAddToGO *<specifier>*
Where *<specifier>* can be any one of the following:

cone <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<sides> <base-radius> <height>
Create cone of varying height and radius

cyl <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<sides> <radius> <height>
Create cylinder of varying height and radius

box <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<height> <width> <depth>
Create box with different height, width, and depth

cf <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<sides> <base-radius> <top-radius> <height>
Create irregular cylinder, which has different radius values for each end

disk <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<sides> <radius>
Create flat disk with a given radius.

circle <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<sides> <radius> <line width>
Create circle formed by line, with given radius

circleD <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<num-points> <radius> <line width>
Create circle made of dashed lines

rect <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<width> <height>
Create flat rectangle

tri <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<X1> <Y1> <X2> <Y2> <X3> <Y3>
Create flat triangle, possibly irregular, using three points.

rTri <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<adjacent-length> <opp-length>
Create flat right triangle by giving opposite and adjacent sides

line <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<X1> <Y1> <Z1> <X1> <Y2> <Z2>
Create one line. Not dynamically updated or associated with other objects.

point <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<X> <Y> <Z>
Create point

quad <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<X1> <Y1> <X2> <Y2> <X3> <Y3> <X4> <Y4>
Create quadrilateral polygon - in the plane

squad <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<a> <b> <d> <c> <n1> <n2>
<u_min> <u_max> <u_step>
<v_min> <v_max> <v_step>
Create superquadric shape - based on forms of ellipses - includes spheres, eggs and toruses

sq <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<size>
Create square plane

wedge <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<num-points> <angle> <sides>
Create wedge

arc <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<npts> <angle> <radius> <line width>
Create arc, with rounded edge line

arcD <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<npts> <angle> <radius> <line width>
Create dashed arc with dashed edge. The argument npts gives the number of points used from 0.0 to your specified angle. The more points, the smoother its rounded edge appears.

text3d <name> <X> <Y> <Z> <yaw> <pitch> <roll> <complexity> <scale> <textstring...>
Based on Inventor 3D text, builds a text string in 3D so that it can be immersed in the scene. The complexity inicates how smooth/blocky the letters are and ranges from 0.0 to 1.0. A complexity of 0.3 is suggested to reduce polygon load in rendering scenes. The scale must be a positive number, and indicates the scale of the text. The final arguments are the text string itself.

imagescreen <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<rgb-imagefile> <screenwidth>
Build a rectangular screen of a given width in meters. The height is determined by the aspect ratio of the image itself to the width you provide so that images appear undistorted. The image shows on both sides of the screen, and the depth of the screen is a small ratio to the given width. The image should be an SGI rgb format image file. SGI Utilities such as *fromgif, fromppm,* and *fromsun* exist to get your favorite image into the rgb format.

sphere <name> <X> <Y> <Z> <yaw> <pitch> <roll> <radius> <complexity>
Buiild polygonal sphere.

wSphere <name> <X> <Y> <Z> <yaw> <pitch> <roll>
<npts> <num-vert> <num-horiz>
<radius> <vert-width> <horiz-width>
Build wire sphere, such as lat/lon sphere used on earth

vrLink <from-dcs-name> <to-dcs-name>
Builds a dynamic link (line) from one dcs (dynamic coordinate system) to another. This link is parented under the from-dcs portion of the scene graph, so it will obey switches above it in the scene graph. Whenever either dcs moves, the link is updated. This is most useful when showing associations between objects in the virtual world (currenlty the link is always red, soon arguments will be available for rgb to set unique colors, for line width).

vrMovie <participant> <moviename> <moviecommand>
Control a moview (video sequence) in Vista, where participant could be "all" or a specific username, moviename is the name of the movie to be played, and moviecommand could be one of the following:

create <movieformat> <startframe> <endframe> <numofscreens>
The "create" command creates the movie given the movieformat, starting and ending frames and the num of screens.

transform <screenNum> <X> <Y> <Z> <H> <P> <R>
The "transform" command is used for applying a translation and rotation toa particular screen. <X>, <Y>, <Z> is the translation along X, Y and Z axisrespectively. <H>, <P>, <R> is the rotaion along Z, X and Y axis respectively.

assign <screenNum>
The "assign" command puts a movie on a particular screen given by its number.

attach <screenNum> <parent>
The "attach" command attaches the screen geometry to a parent node in the scene.

start
The "start" command starts the movie, it can be stopped and resumed.

stop
The "stop" command stops the movie. It can be resumed.

pause
The "pause" command pauses a movie, much like stop.

resume
The "resume" command resumes a movie.

reverse
The "reverse" command reverses the movie, i.e., plays it backwards.

remove
The "remove" command removes a MoviePlayer and the screens and moviesassociated with it.

1.3.2. Object Deletion

Deleting an object also causes any swich or dcs nodes that were created with it to be removed unless they have other children not associated with the node named for deletion.

vrDelGO <obj-name>
Delete a named graphic object from the scene graph

vrDelSimObj <sim-type> <sim-obj-name>
Delete a named simulation object and all its attendant structures. Currently works for satellite simulation objects.

vrDelink <from-dcs-name> <to-dcs-name>
A dynamic link is specified by it from-to association. By sending a vrDelink message the the same dcs names as an existing link, you can force it to be deleted (un-linked).

1.3.3. Object Modification

Object transformation is the movement, rotation, and scaling of visual objects in a three-dimensional setting. We also include changes to parameters of the object here, such as color or transparency, complexity, or dimensions.

vrTranslateGO <obj-name> <x> <y> <z>
Translate an object to a given point. Translation is local to frame of reference, and not in world coordinates.

vrRotateGO <obj-name> <yaw> <pitch> <roll>
Rotate named object. Angles are in degrees.

vrScaleGO <obj-name> <scale-num>
Scale named object. Range is 0.0,100.0

vrColorGO <obj-name> <red> <green> <blue>
Set the color of a named Graphic Object. Will also (Performer 2.0 version) set color for all geometry in the scene graph under the graphic object, which means it can be used to change the color of graphic objects loaded as model files.

vrSetMaterial <node-name> <materialType> <red> <green> <blue>
(Performer 2.0 version) Set the material values for a named node in the scene graph. Will also set material values for all geometry in the scene graph under the scene graph node, which means it can be used to change the material data of graphic objects loaded as model files. Note there can be un-expected color changes in other related objects, if materials are shared after scene graph optimization. To force the changes to be local to geometry under a given node in the scene graph, change its material to be unique (different from others so it is not merged in optimization). Note that as the number of unique materials goes up, the drawing efficiency can go down, so be sure to define unique materials only if you'll need to change the material settings dynamically. The material Type can be one of four string tokens: diffuse, ambient, specular, emission. The diffuse is typically what you want to just change the color of an object.

vrDCSData <participant> <obj-name>
This message tells the Vista Viewer tor a given participant to find the named DCS and report back information on its position and orientation and scale. It sends out the TScript message vrcDCSData in response to this query.

vrcDCSData <obj-name> <x> <y> <z> <yaw> <pitch> <roll> <scale>
Result returned after a vrDCSData query. This gives information about position and orientation for a given named node's transformation matrix (dcs). So if your process sends a vrDCSData Tscript message, it should also register interest for the return vrcDCSData message.

vrBSphereData <participant> <node-name>
Accepted by the Vista Viewer (Performer 2.0) this message requests bounding sphere information for a named node in the scene graph associated with a given participant.

vrcBSphereData <participant> <node-name> <world-x> <world-y <world-z> <radius>
Return result from a vrBSphereData query, which has the world position, followed by the radius in meters of a bounding sphere which encloses all the scene graph elements below the named node.

vrBBoxData <participant> <node-name>
Accepted by the Vista Viewer (Performer 2.0) this message requests bounding box information for a named node in the scene graph associated with a given participant.

vrcBBoxData <participant> <node-name> <world-x> <world-y <world-z>
<min-x> <min-y> <min-z> <max-x>> <max-y> <max-z>
Return result from a vrBBoxData query, which has world position, followed by the minimum xyz and maximum xyz defining a bounding box which contains all the scene graph elements below the named node.

vrLock <dcs-name> <toggle-val-0-or-1>
Lock an object into place, so that it cannot be transformed or oriented differently

vrRollGO <obj-name> <roll-threshold>
Set threshold for roll whenever this object is traveling on a path. The roll threshold is in degrees, and if it is given a 0.0 argument, the object will not roll (bank) at all as it turns from one path segment to another.

vrConstrain* <dcs-name>
Set the constraints for a named transformation matrix (DCS). These can be constrained to an upper or lower limit, for any axis or orientation.

vrAttachPathGO <obj-name> <path-name>
Takes path argument, and stores this with graphic object so that the object can travel along this path once started.

vrStartPathGO <obj-name>
Similar to vrStartPA, this message tells Vista to start the graphic object along a path, if one has been previously attached to the graphic object.

vrStopPathGO <obj-name>
Similar to vrStopPA, this message stops an object if it is traveling along a path.

1.3.4. Event Monitoring

Tscript messages are used to report the occurrance of events as well as for registering them. Whenever an event involves an object or participant, the specific name of the participant or object is included in the message. Such event monitoring is necessary and useful in a setting where the people (participants) may do almost anything, and where disparate simulations are coming together to update a world model.

vrPick <object-name> <participant> <local-x> <local-y> <local-z>
Report pick event in Vista, pick point is local to the object picked. A vrPick message is sent out from a participant's vista viewer is output whenever the user selects a named object.

vrPOV <participant> <x> <y> <z> <yaw> <pitch> <roll>
Report point of view (POV) of participant XYZ in meters, orientation in degrees. Output from the Vista viewer periodically, according to threshold settings for change in position and orientation.

vrEvent <participant> <event-name> <event-type> <type-args>
Register an event for a given participant.

Event types can be the following for the vrEvent message can be the following:

sphere <dcs-name> <radius>
events are reported when other dcs center points are detected to be inside of the given sphere
spherecol <dcs-name>
events are reported when the sphere of another dcs is detected to be intersecting with this sphere
frustum participant
events are reported when the bounding spheres of other nodes in the scope are detected to be inside the participant's viewing frustum (Performer 2.0 version requires participant token)
frustum <dcs-name> <near> <far> <vertical-fov> <horizontal-fov>
events are reported for the defined frustum when the spheres of other nodes are detected to be inside this defined frustum (Performer 2.0 version only)
box*
segments*
cylinder* <dcs-name> <axis-vec> <half-length>

The scope for each vrEvent is modified using the vrEventScope message below.

vrEventScope <participant> <event-name> <duration> <scope-objs>
Set the scope for a named event, where duration can be the tokens *single, continuous, none, delete,* or a number indicating seconds since receipt of message for scope. The scope-objs can be any named objects in the environment, such as other participants, or graphic objects.

vrEventNotice <participant> <event-name> <scope-name> <e-type> <e-specs>
Events that are logged for a participant can result in a returned event notice. This gives the name of the registered event, and specifics for that event type describing the event. Output from the Vista Viewer whenever the conditions for a given event are met.

1.3.5. Participant Control

Participants in the Training Studio must at times be guided to assure effective instruction. The TScript primitives which follow help in the task of participant guidance. Note: in TScript messages which take a participant argument, this argument is currently their user name that they have logged in with. The special participant *all* can be used as a wild-card to send messages to all the participants at the same time.

vrPartMoveRel <participant> <dcs-name> <radius>
Move immersed participant near a transformation matrix <dcs-name> based on their current orientation - their view is moved so that the object is in the center of their view. Related to the spherical move.

vrPartSphereMove <participant> <object-dcs-name> <radius>
Tether an immersed participant to a named transformation matrix <dcs-name> a given radius away (in meters, scaled according to the tethered object). Effect is that immersed person can look up to see under object, look down to see top of object, and in general move their view in a sphere around an arbitrary moving or stationary object.

vrPartTrack <participant> <object-dcs-name>
Accepted by the Vista Viewer for a given participant. Causes orientation of the viewer to continuously change in order to follow a focus object. The participant can move around, but they stay focused onto the object. This viewing mode is like the focus for a path, but the user is free to change their position. (replaces vrLookAt in earlier versions).

vrPartPosition <participant> <world-x> <world-y> <world-z>
Accepted by the Vista Viewer, this sets a given participant's position in world coordinates. Replaces vrMoveTo, which doesn't differentiate between participants.

vrTether* <participant> <object-dcs-name>
Accepted by the Vista Viewer to tether a participant to a given named object. As object moves, so does the participant, though they can move relative to the tethered object.

vrRayLength <participant> <ray-len-meters>
Sent out and accepted by the Vista Viewer, this information indicates a participants ray length, and is used to update the ray representations in other participants viewers.

vrNearClip <participant> <far-clip-dist>
Set the participant's near clipping distance, in meters.

vrFarClip <participant> <far-clip-dist>
Set the participant's far clipping distance, in meters.

vrFov <participant> <horizontal-deg> <vertical-deg>
Set the participant's horizontal and vertical field of view, measured in degrees..

vrIpd <participant> <ipd-val-meters>
Accepted by the Vista Viewer, this sets the inter-pupillary distance for a given participant. It is in meters, so meaningful numbers should be 0.05, etc. for 5 centimeters. This value is used to separate the software channels in Performer by the given amount, so that views have a slightly different perspective, such as your eyes experience.

vrSkyModel <participant> <sky-model-number>
Set a given participant's sky model. Currently in range 0,5, where 0 is to clear to black such as is used for the orbital mechanics demonstration's black sky.

vrText <participant> <parent> <textname> <style> <size> <red> <green> <blue> <msgstring...>
Accepted by the Vista Viewer (Performer 2.0), this message creates or modifies the named text node in the 3D scene. The text displayed can have a style attribute of flat (2D polygons) extruded (3D) or textured (as an image).

vrSetText <participant> <textname> <msgstring...>
Accepted by the Vista Viewer (Performer 2.0), this message modifies the text string for already existing text in the 3D scene (created previouusly using vrText).

vrScreenText <participant> <text-bin-num> <text-string...>
Accepted by the Vista Viewer to display overlay text onto a given participant's view. There are currently 5 text bins. Bin 2 is reserved for displaying the mode that the participant is in, in the upper left corner. The screen text resulting from this message will always be in the middle of the screen, and starting from the left side.

vrScreenTextRel <participant> <text-bin-num> <rel-x> <rel-y> <text-string...>
Accepted by the Vista Viewer to display text relative to the screen. Similar to vrScreenText, except rel-x and rel-y arguments specify where on the screen the text appears. The origin is the lower left, and the range is (0.0,1.0). e.g. sending rel-x = 0.1 and rel-y = 0.5 will place your text on the left side, near the middle of the screen vertically.

vrScreenTextClear <participant>
Accepted by the Vista Viewer, this message will cause it to clear the screen of all text. If you desire just one text-bin to be cleared, send it a vrScreenText message for that text bin with an empty screen.

vrSetThreshold <participant> <thresh-type> <threshhold-value>
Accepted by the Vista Viewer, this message will set a type of threshhold for a given participant. The types can at present be: *translate* or *rotate*, and again these are in meters and degrees, respectively.

vrBldPA <path-name>
Build a path container using given name

vrAddToPA <path-seg-type> <path-name> <path-seg-parameters>
Add path segment to a named path, with specific path segment parameters

Path segment types and their respective parameters follow:

speed <path-name> <desired-speed> <rate>
Set the speed of travel for all subsequent path segments. Overridden by any following speed segments.

delay <path-name> <seconds-delay>
Set a delay segment into path. Object or participant will wait at endpoint of previous segment until this delay is expired.

fillet <path-name> <turn-radius>

Specify smooth turn between previous path line segment and next path line segment. Turn radius argument is given in meters.

arc <path-name> <pivot-x> <pivot-y> <pivot-z> <radius> <angle1> <angle2>
Specify explicity arc around a given pivot point in world coordinates. The radius is in degrees. The first angle modifies the object's yaw, the send angle modifies it's roll as it progresses through the arc.

line <path-name> <x1> <y1> <z1> <x2> <y2> <z2>
Specify path segment where object or participant is oriented to look down the segment as the travel it. The arguments are world coordinates for start and finish points of the line.

lineOO <object1-name> <object2-name> <x1> <y1> <z1> <x2> <y2> <z2>
Specify path segment that points object or participant orientation at object1 to start with, and finishes pointing at object2. The other arguments are the world coordinates for a start and end point for the path segment.

lineDD* *not enabled*

lineDO* *not enabled*

vrSelectPA <path-name>
Select path name for further use (soon to be phased out - redundant)

vrStartPA <participant>
Handled by the Vista Viewer for a given participant. Start previously selected path (soon to take path-name argument and obsolet vrSelectPA)

vrStopPA <participant>
This message is handled by the Vista Viewer for a given participant. Stop any path currently in progress.

vrNewText* <text-name> <double-quoted text>
Send text for Vista Viewers to display

vrMakeLabel* <object-dcs-name> <double-quoted text>
Send text to Vista Viewer to have it display moving text label

vrMoveTo <x> <y> <z>
Move participant to a given place. (Soon to take participant name as argument)

vrLookAt <obj-name>
Set participant's orientation (without moving them) to look at a given object. (Soon to take participant name as argument)

1.3.6. Scene Control

vrSwitchGO <obj-name> <switch-num>
Set switch to 0 to turn off objects in scene, 1 to turn on

vrAddSwitch <parent-name> <switch-name>

Add a switch node to the scene graph - useful if need to add objects below

vrAddDCS <parent-name> <dcs-matrix-name>
Add a transformation matrix to scene graph

vrAddGroup <parent-name> <group-node-name>
Add a group node to scene graph - used to collect children together

vrFOV <participant> <field-of-view-value>
Set the field of view for a participant range in Derek 0,90

vrReparent <new-parent-name> <node-name>
Set a named node in the scene graph to have a new parent

vrLodScale <new-lod-scale>
Set the scale for level of detail transitions, range 0.0,100.0

vrFindParent <named-node>
Vista viewer will respond to this query with a message giving parent of named node in the scene graph. Useful for traveling up the scene graph from a picked object to affect a wider collection of objects.

vrcParent <parent-name>
Output by the Vista Viewer in response to a *vrFindParent* query.

1.3.7. Session Control

vrAllCall
Requests all components connected to communication bus to identify themselves by responding with a vrComponent message

vrComponent <component-name>
Response to all call request

vrStop <component-name>
Stop execution for named component, if *all* is used, all will respond

vrStopSave <component-name>
Stop after saving state if capable of saving state

vrDebugMsgs <toggle-val-1-or-0>
Accepted by the Vista Viewer, this message will toggle the printing of all incoming TScript messages to standard output. Very useful if you are building a utility that sends TScript commands to Vista, and you need to be certain they are formed correctly (according to this user guide section).

vrToggleSim
Accepted by the Vista Viewer , globally toggle on or off the whole simulation in Vista

vrSimRate <seconds-multiplier>

Accepted by Vista to set the rate for the global simulation.

vrClockNew <clock-name>
Create a new simulation rate clock

vrClockRate <clock-name> <rate>
Set the rate of a named simulation clock

vrClockToggle <clock-name> <tog-val-0-or-1>
Toggle on or off an individual, named simulation clock.

vrClockSet <clock-name> <new-time>
Set the time value for a named simulation clock

Appendix B: VET Communications in V-RIDES

V-RIDES provides support for sending and receiving TScript messages via Tooltalk. Sending is straightforward. A **TScriptSend** V-RIDES function call is used to send a TScript message. The first parameter of **TScriptSend** specifies the TScript type of the message; the second parameter specifies the content of the message.

For example,

```
TScriptSend ("vrTranslateGO", "cyl1ptr 0 -0.75 -0.1");
```

sends a message of the vrTranslateGO type that has the effect of moving the virtual environment graphical object *cyl1ptr* to the location [0, -0.75, -0.1]. If the object was previously at [0, -0.75, 0], then it moves down (in the Z dimension) 0.1 meter.

To make a V-RIDES simulation receive TScript messages, an author must ensure that an event establishes what V-RIDES *text attribute* is to receive messages of a particular TScript type. When a new Tooltalk message that has been linked to an attribute is received, the attribute's value contains the name of the message type plus the contents of the message.

For example,

```
"vrPick cyl1ptr_grf_dcs allen 0.010958 -0.246440 0.253525"
```

specifies that there was a vrPick event on the *cyl1ptr* object at the point identified by the three numbers (in the coordinate system of the object).

To specify that a particular V-RIDES attribute (say, one named *vrPickInfo*) should get vrPick messages, the author would use this function call in an event:

```
TScriptLink ("vrPick", .Stage.vrPickInfo)
```

where the *vrPickInfo* V-RIDES attribute belongs to a scene object named *Stage*.

Note that once a V-RIDES attribute has been *linked* to a particular message type it will **automatically** receive all broadcast messages of that type. It is not necessary that the author write an event to monitor for Tooltalk messages of interest. The act of establishing the link using TScriptLink() is enough to ensure that V-RIDES will catch the message and direct it to the appropriate V-RIDES attribute.

It is possible to make more than one message type 'report to' the same V-RIDES attribute. Therefore, an author could arrange to have all messages of interest sent to one attribute, and could then respond to these messages using V-RIDES events and relation rules. The message type of a message is included in the value placed in the receiving attribute in order to support this approach. Alternatively, of course, an author can assign each message type of interest its own V-RIDES attribute, in which case the author's rules can simply ignore the first field (the message type field) in received messages.

See the descriptions of the language extensions below.

Core VET Communications Functions for V-RIDES

These are the extensions to the V-RIDES rule language in support of V-RIDES communications capabilities.

TScriptSend (<text1>, <text2>)

where <text1> is a TScript message type, such as "vrInterestIn", and <text2> is a text string with all the appropriate space-delimited arguments for messages of the type specified by <text1>.

TScriptLink (<text>, <textAttributeReference>)

where <text> is a specification of a TScript message type (such as "vrPick"), and the second argument is a reference to a text attribute. After this command has been issued, TScript messages of the specified type will be deposited in the specified text attribute.

In addition, two new V-RIDES functions are provided to assist the author in accessing space-delimited 'fields' in a string, such as the TScript message strings that are automatically assigned to linked attributes:

NumFields (<text>)

This function returns the number of white-space-delimited fields in its string argument.

GetField(<text>, <number>)

This function returns a text value that is the *i*th space-delimited field specified by the number argument, *i*. If there is an error condition (e.g., the text argument is empty, or there is no *i*th field), then the function returns "", the empty text string.

V-RIDES Support for Attribute Serving

The following features are of interest only to authors using the Sun version of V-RIDES. While it is possible to author PC versions of V-RIDES that include rules with the new TScript functions, the underlying Tooltalk-based communications bus is not supported on PC versions of Unix, so the PC (SCO/Linux) version of V-RIDES cannot actually be used to communicate with other TScript-aware applications.

V-RIDES can now handle TScript messages of the form:

vrRqstReg ridesname .scene.attribute number

This is used to register an interest in the V-RIDES attribute, such as **.scene.attribute**. All parameters but the attribute name are ignored in this version, but future versions may support the presence of more than one instance of V-RIDES in the training environment.

vrRqstIgnore ridesname .scene.attribute

This is used to "de-register" an interest in an attribute that was previously registered with a vrRqstReg message.

Symbols, Abbreviations, and Acronyms

Used to explain the meaning of symbols, abbreviations, and acronyms; need if there are more than 5 not readily recognized as standard. Must be displayed in two columns with the abbreviations and acronyms listed in alphabetical order and aligned with the left margin. Each entry begins on a new line, followed in the second column by its definition.

ARPA	Advanced Research Projects Agency
BTL	Behavioral Technologies Laboratories, located in Redondo Beach, CA, a performing organization in the Lockheed Martin VET effort, a laboratory of the University of Southern California.
DIS	Distributed Interactive Simulation, a real-time distributed message protocol used in training and operational simulations developed by ARPA and now an International Standards Organization standard.
FBM	Fleet Ballistic Missiles program, a Lockheed Martin program funded by the U.S. Navy for the production of submarine-launched ballistic nuclear missiles
HPAC	High Pressure Air Compressor, an oil-free air compressing system prevalent on many navy vessels, which prepares compressed air for gas turbine engines.
ICAI	Intelligent Computer Aided Instruction, a method of instruction whereby an intelligent model of a student's understanding is used to guide a student during instruction using a computer.
I/ITSEC	Interservice/Industry Training Systems & Education Conference
IPEM	Integrated Planning, Execution and Monitoring architecture for coordinating different planning strategies as required for SOAR activities.
ISI	Information Sciences Institute in Marina Del Rey, CA, a performing organization in the Lockheed Martin VET effort, affiliated with the University of Southern California in Los Angeles, CA.
JPL	Jet Propulsion Laboratories, a National Laboratory affiliated with the National Aeronautics and Space Administration.
MCO	Multi-Channel Option for Silicon Graphics Onyx Workstations, a necessary option to provide separate video channels used in immersive virtual environment displays.
ONR	Office of Naval Research, the funding agency for the VET effort.
RIDES	Rapid Instructional Development for Educational Simulation, a software system for authoring simulation-based training developed at BTL
SGI	Silicon Graphics Incorporated, a workstation company whose whole culture centers around fast 3D graphics.
SOAR	A platform independent, cognitive architecture based on a production system which seeks to address those capabilities required of a general intelligent agent.
STEVE	SOAR Training Expert for Virtual Environments
Tcl/Tk	A windowing interface toolkit assembled around a unix-shell like interpreter originally developed at UC Berkeley.
TScript	Training Script message protocol for virtual environments, a distributed message protocol analogous to DIS PDUs or Xlib protocol for X-windows.
URL	Uniform resource locator, a tag that indicates a media format and location on the Internet as part of the World Wide Web.
USC	The University of Southern California in Los Angeles.
VE	Virtual Environment, a 3D visual display and accompanying simulation which represent some aspect of an environment. Expanded forms of VE also address other senses such as audio, touch, etc.

The following types of messages will be automatically sent by V-RIDES as a result of the registration of attribute interest by another application, such as Soar.

vrAttrChange/rides/.scene.attribute 47.2

In this release, all V-RIDES simulations are known only as "rides". In the future we will probably send the simulation name in place of "rides".

vrAttGone/rides/.scene.attribute

This message is sent if a V-RIDES author deletes an attribute that is currently being observed by the collaborating application (e.g., Soar).

vrErrorAttr rides .scene.attribute unknown

This message is returned if the collaborating application tries to register interest in a non-existent attribute.

High-level coordination

vrAttStart rides
vrAttEnd rides

These messages are used to synchronize with a collaborating application. In theory, it would be possible to broadcast an inconsistent set of simulation data if data were not synchronized. V-RIDES therefore completes a 'simulation loop', sends a vrAttStart message, sends the attribute-value pairs of interest, and then sends the vrAttEnd message. The receiving application can confidently make use of the received data once the vrAttEnd has been heard.

V-RIDES also announces its presence to collaborating applications. It sends

vrProcBegin rides

when it joins the Tooltalk bus, and

vrProcEnd rides

when it leaves.

V-RIDES also responds automatically to vrAllCall messages with the TScript message

vrComponent rides

VET	Virtual Environments for Training, a Defense Department focused research initiative concerned with applying virtual environment technology to training
Vista	Distributed virtual environment display and interaction system developed by Lockheed Martin for training purposes.
VR	Virtual Reality *see Virtual Environment*
V-RIDES	Virtual Rapid Instructional Development for Educational Simulation. A special version of the RIDES program for use in developing simulations and tutorials that collaborate with Vista Viewer and Soar to deliver training in virtual environments.
VRML	Virtual Reality Modeling Language, an analog to HTML used for documents, but focused on 3D objects and scenes for the World Wide Web.
WWW	World-Wide Web, a system incorporating the HTTP message protocol and the HTML document description language that allows global hypertext over the Internet.